Query Prediction in Large Scale Data Intensive Event Stream Analysis Systems

Song Huaiming[1,2] Wang Yang[1,2] An Mingyuan[1,2]
Wang Weiping[1] Sun Ninghui[1]
1. Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences
2. Graduate University of Chinese Academy of Sciences
Beijing, China
{shm,aaron,anmingyuan,wpwang,snh}@ncic.ac.cn

***Abstract*—Hot-spot events accessing has recently received considerable attentions in the event stream historical analysis systems. Noting that predicates in SQL(Structured Query Language) requests usually have similarity features in a short time in event stream systems, that means events frequently queried recently might be queried again in the near future. This paper proposes a prediction model to forecast query predicates and then to choose them for speculative execution. We propose an adaptive two-level scoring (TLS) prediction algorithm, which can adjust parameters according to the system resource usage conditions. We introduce two metrics accuracy rate and efficiency rate, for query prediction evaluation, and make a detailed analysis of system costs. Our experimental results in DBroker system demonstrate the TLS algorithm and local speculative execution method can significantly reduce query response time.**

Keywords-query prediction; event stream; stream database; DBroker; TLS

I. INTRODUCTION

Data stream management systems [1,5,10,11,14] are widely used in a range of domains like financial data processing, network monitoring, communication data records management and sensor networks. Event stream system is a special kind of data streaming systems in which event is a basic element of the stream. An event is a piece of data that means something happened or had some status at some time in the real world. Event e is always represented as $e=\{p, t\}$, in which t is a time stamp of event e and p is a set of attributes values. Event stream can be looked as a totally ordered set of time serials $\{\ldots,e_{t-1},e_t,e_{t+1},\ldots\}$. Events are continuously generating and arriving at the system, so the stream is an endless series. Now it is always to store historical data of the series in database systems for rich semantic off-line analysis [5,12,33], as is called event streaming database. Compared to traditional databases, event streaming database has much difference, and the difference between them is showed in Table I.

Unlike continuous queries for real-time stream query processing, an event streaming database system uses SQL for historical data analyzing. An example of query request is that:

SELECT *select_clause* FROM *event_base*
WHERE *event_attributeX= value*
AND *event_time* BETWEEN (t_1,t_2)
[GROUP BY *group_by_clause*]
[ORDER BY *order_by_clause*];

TABLE I. DIFFERENCE BETWEEN TRADITIONAL AND STREAMING DATABASES

	Traditional database	**Streaming database**
Stored records	relatively complex relations	simple event serial
Data period	only current state	aging problem
DML operation	insert, update, delete	batch insert and delete
Insertion rate	relatively low	very high
Query results	precise query	precise and approximate

Coarse-grained locality is common in event stream historical analysis systems: some events with certain attributes values would be frequently accessed in a short period, we call these events hot-spot events. Hot-spot events are dynamically drifting as time elapsing. In an event stream system, coarse-grained locality mainly behaves as time conditions locality and attribute values locality.

Time conditions locality: user requests always focuses on events data in a small range of time interval. In the above instance, time interval (t_1,t_2) is always a recent period of time, that's because the newest data is always the most important in event stream systems.

Attribute value locality: data accessing in event steaming database focuses on some specific events in a small period of time, eg. "*event_attributeX = value*". Events with certain attribute values frequently queried recently might be queried again soon in near future. We call events with these attributes values hot-spot events. Hot-spot events would be changed when user's interests change, that is called hot-spot drifting.

According to the two kinds of locality, query prediction could be carried out from following two aspects:(i)query time conditions prediction, e.g. to predict t_1 and t_2 values in later queries, fortunately, time interval seems to be equal length and mostly in a recent period in the same query pattern. (ii) hot-spot events prediction, to predict attribute name and values in later queries, a common idea is that the events with same attribute values will be queried again soon in the future if they are frequently queried in a short time.

978-0-7695-3449-7/08 $25.00 © 2008 IEEE
DOI 10.1109/GCC.2008.115

This paper analyzes the real query workloads of DBroker system, a large scale data intensive system based on event stream application, and then introduces a system design of query prediction model. We also propose a two-level scoring(TLS) prediction algorithm and carried out a set of detailed experiments in DBroker systems, our experimental result shows that our prediction approach makes a significantly reduce of query response time.

We have made following contributions in this paper: (1) We make a detailed analysis of DBroker system, an event stream analysis system on Dawning 4000L Cluster platform, to illustrate the hot-spot events accessing problems. (2) We propose a query prediction model, including query request analyzing, an adaptive TLS prediction algorithm, and query cost evaluation. (3) We propose two speculative execution methods, global and local methods, and make a comparison of them.

The rest of this paper is organized as follows. Section 2 presents an overview of related work. Section 3 introduces a real event stream management system DBroker, and then makes a query workload analysis. In Section 4 we present the query prediction model and the two-level scoring (TLS) algorithm as well as speculative execution methods, while Section 5 gives a system costs analysis of the prediction model, and the adaptive improvement of TLS method. Section 6 presents experimental results. Section 7 discusses the challenges of cluster environment and the limitations, and Section 8 summarizes the prediction model and points out our future improvements.

II. RELATED WORK

A. Event Streaming Database

New applications that must deal with vast numbers of event streams are becoming common in recent years. Most researchers have focused on streaming online continuous query processing [6,13,24,31], especially on streaming aggregation and join[4,16,18,20,29]. However, streaming online processing can't meet all the requirements of applied systems, they need further offline analysis with much richer semantic. A class of large scale data intensive applications, like sensor data processing and network intrusion detecting systems, employ databases to store and analyze those historical data, and it is always called streaming database[33]. Comparing with traditional database systems, streaming databases pose new challenges in data management and query processing.

Data management: including data organizing, placement, partitioning and indexing. Because of infinite nature of event streaming, but system storage size is limited, event stream database can not store all the historical data. Old data would be deleted from database or be compressed, replacing with statistical data [32] or sample data. Data writes to system in an append-only pattern, once written, it will not be modified. Indexing is also a difficult problem, because streaming insertion rate is very high, and also there is data aging problem [3].

Query processing: including data accessing pattern and query answering. Data accessing frequency is not unified in event stream database, as all data have its life cycle: the newest data is the most important and then most often to be queried in the system[3,5,13]. Query results may not be required as precise as traditional database, instead of approximation, or it would make times of efforts to achieve only a little more precision, that may not be practical in a large scale system.

In an event streaming database, indexes and material views at table level are not suitable. There are three reasons. First data continuously keeps arriving at a very high rate so it is expensive to create indexes and views for new coming data. Second event streaming data has it own life cycle and it will be seldom accessed when it is getting old, that means indexes and views for old data are useless. Third the storage for indexes and views is too large and introduces a lot of system costs. The high maintenance costs and the low utility rate make it is too expensive to use indexes and views, and it can't be counterweighed by benefits getting from queries. In a word, indexes and material views common used in traditional database [34] are not ideal solutions for query optimization in large scale data intensive event steam system. Roxana[35] proposed on-demand views to support high insertion throughput in network intrusion detection system, but it is based on rule matching, e.g. special events occur. Sai[36] proposed partial index instead of index all data, and the tuple-level index offered better search and query efficiency than table level index in distribute systems. Our prediction model doesn't make use of index or material view, and it automatically specifies hot-spot events by prediction algorithm and then employs speculative execution for query optimization.

B. Locality and Caching

Time conditions locality is an important feature in event stream systems[23]. In an event stream system, data accessing frequency is shrinking with time elapsing, user requests always focus on streaming data in a newest short time interval, usually some recent slide windows[5,6,13]. Altiparmak[3] pointed out this kind of data aging problem, and proposed incremental quantization techniques to enable aging in an efficient manner. Zhang[32] proposed a multiple levels of temporal granularity to store historical data: older data is aggregated using coarser granularity while more recent data is aggregated with finer detail. All the research pointed out the data aging problem in streaming database, and focused on data management and query processing techniques in streaming systems, without considering attribute values locality of streaming data.

Semantic cache is an efficient way to reduce query response time, because of hot-spot accessing characteristic in information retrieval systems, web search engines and mobile environment. According to granularity of the units cached mentioned in [17,27], caching schemes can be classified into page level caching, query level caching and table level caching. The page level caching is so common used in storage subsystems, and the query level caching makes benefit from reusing query results, while the table level caching is a way that caches entire objects of the table, e.g. indexes and material views. Query result caching have attracted much attention in several research projects [9,17,25,27]. Some of them cache entire result as a whole, and others cache chunks for fine granularity caching. The former benefits when a new query is

subsumed by a previously cached query, the latter one also benefits in overlapping situations, by partially reusing of query results. However, semantic cache is not suitable for event stream systems, in which streaming data continuously changes all the time, and the query results are much different in different slide windows.

C. *Prediction and Speculation*

Predicting the future events based on historical data is useful and applicable for proactive dynamic query processing, events monitoring systems and decision support systems. Reference [2,15,22,28] attempt to predict the behavior of query execution and plan ahead for possible contingencies. Duan [19] described a system called Fa that supports declarative forecasting queries with accuracy estimates. Gooijer[21] made a comprehensive review of time series forecasting over the past 25 years. Our work differs from these works that we make prediction on user requests operations series, other than on execution behavior in dynamic plans, or on approximate results for making decision.

Speculative execution is an effective way to reduce user response time if proper used, and it has been much studied in database and information retrieval systems. Barish[7,8] employed speculative execution to information gathering plans, to increase the parallel degree of system. In information gathering plans, operation series always follow a regular way, and information fetching from remote sites is an inefficient operation with high delay, so it leads to much of local system resource in idle. In this situation, speculative execution makes benefits from exploring system parallel degree. But it is different in event stream systems, because in information gathering plans, extra costs introduced for remote sites are not considered, but in our prediction model it is an important evaluation criteria, which can not be ignored. Polyzotis [26] applied speculative execution in interactive query processing, it takes advantages of the user input time (user thinking time) and launches partial query before users total request submitted. It is not suitable in many large scale data intensive systems based on event stream application, as such users input time is too small to be considered comparing to query processing time. Wang [30] focused on prediction subsystem with conditions on streaming time series, and then reacted with certain actions. Such kind of speculation is based on approximate evaluation for the conditions, however it is not the situation discussed in this paper, either.

III. DBROKER SYSTEM

DBroker is an event stream management system on Dawning 4000L cluster. It supplies both special and general services and strategies for event stream data analysis, including data loading and placement, query processing and result merging, statistical analysis, configuration consistency and service management. It is a hybrid structured parallel database system used for network intrusion detecting historical analysis. The main system architecture of DBroker is showed in Fig. 1 and this kind of architecture can well meet the requirements of scalability, manageability and availability.

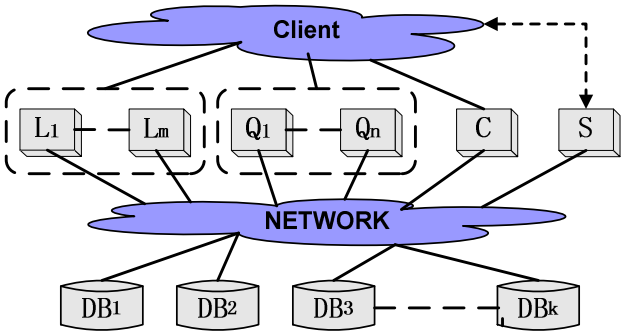

Figure 1. System architecture of DBroker.

In Fig. 1, Ls represent for load server nodes, used for streaming data partitioning and inserting, Qs represent for query server nodes, used for query processing, DBs represent for database nodes, C and S are used for service management. Among these nodes, Ls, Qs, C and S are called server nodes and work as mid-ware of event steaming database, which provide data accessing and processing entry for the whole system, while DBs provide for data storage and management.

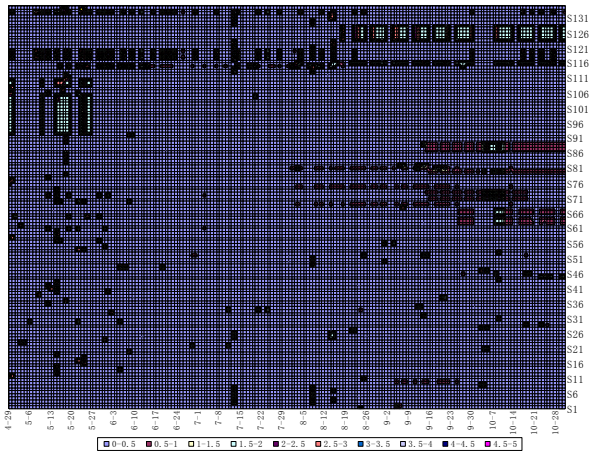

Figure 2. Event accessing statistics in DBroker system (2006.4~2006.10).

DBroker is one of typical instances for large scale data intensive system based on event stream application. It is back-end of a network intrusion detection system, used for off-line analysis. We have made a detailed study of system query trace, and we found that events accessing frequency is not uniform in the system. Fig. 2 shows the event accessing statistics in about 6 months(2006.4~2006.10) from system running log. In Fig. 2, X axis represents for time, Y axis represents for different event attribute values, and the color represents for logarithm of frequency. It contains majority of all query requests, about 86.27% of all in the 6 months. From the figure we can see that event data accessing has obvious hot-spot characteristic. Queries are always focus on events with some certain attribute values in a small period of time, and the hot spots may be drifting as time elapses. Our query prediction model is to predict those hot-spot events, by analyzing table-scan and filter operations in user SQL requests.

IV. QUERY PREDICTION

A. Prediction Model

We have developed a prediction model in DBroker system to predict users following requests, for query optimizing. Our prediction model consists of three units: query request analyzer, operation scoring unit and query speculative execution unit. Query analyzer is used for SQL analysis and dividing a SQL request into query operations; operation scoring unit maintains a set of scores of the operations generated by the query analyzer and then predicts which operations might come in the new future; query speculative execution unit is used for speculative execution of those operations predicted by scoring unit. As is showed in Fig. 3, when a new user request comes, query analyzer first breaks it into an operation tree, and then finds out whether there is any operation hit in the prediction pool, if hit and result of the operation has been prepared by the speculative execution unit, then query server will reconstruct execution plan, to make use of the prepared result, thus query response time would be dramatically reduced, if not hit query server will execute the request in a normal way.

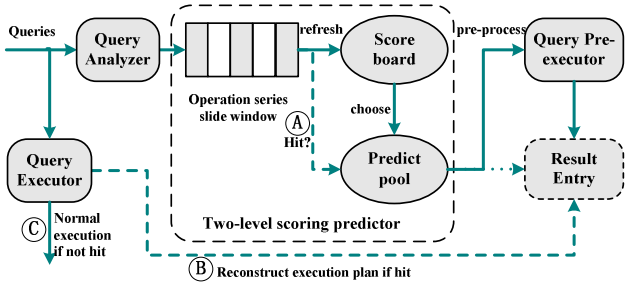

Figure 3. The structure of prediction model

In a query execution plan, hot-spot events are always reflected as table-scan and filter operations, so we mainly take account of predicates in where clause of user SQL requests. We introduce a two-level scoring (TLS) algorithm for hot-spot events prediction, and the two levels consist of score board and prediction pool. An operation consists of an event attribute and its value, just like "*event_attributeX= value*". The score board keeps all the scores of operations, and if an operation comes, then its score will be added with a bonus, otherwise its score will be decreased with a penalty. So operations with high score in score board means that they keep coming frequently and more likely to come in next time window. The prediction pool is a set of operations with high scores in the score board. First of all, we introduce two important metrics of prediction model, accuracy rate and efficiency rate, and detailed working mechanism will be described in later subsections.

Accuracy rate: among all the user query requests, the percentage of operations hit in prediction pool. The high accuracy rate is, the more request will be hit.

Efficiency rate: among all the operations in the prediction pool, the percentage of them hit in later queries. The high efficiency rate is, the less extra workload will be introduced to the system.

Accuracy rate and efficiency rate are two most important metrics to measure two aspects of prediction model, accuracy rate is a performance metric and efficiency rate is a resource-workload metric. High accuracy rate would dramatically reduce query response time, while high efficiency rate would introduce less extra workload of the system. Detailed analysis of response time and system workload will be discussed in Section 5.

B. Request Analysis

When a user request comes, it will be broken into a set of operations by query analyzer to find out whether the operations hit in the prediction pool or not. Then all user requests flowing through query request analyzer can be looked as operation serials which are ordered by arrival time. Combined with slide window technique, the operation serials will be divided into continuous small windows. All the windows are equal length of time interval. Our prediction algorithm treats all the operations in a window as a whole, and predicts operations might be submitted in the next time window.

We focus on operations in 'where' clause in a query SQL, and there are several reasons. One of the most important is that table-scan and filter operation is a time consuming step in the query plan, so speculative preparing would get dramatically reduction of execution time. Another reason is that scan and filter operations are always at the bottom of execution plan tree, so the operation input can be determined before execution. In a large scale data intensive system, high I/O cost is the bottleneck of performance improvement, so speculative preparing of I/O fetching would make great sense. Query analyzer breaks a SQL request into operations by structure analyzing, and the output operation serials will be the input of the following operation scoring unit.

As operations in a time window are looked as a whole, the size of time window is a key factor to be considered in query prediction model. If it is too large, the number of operations in a time window will be too large, and it is hard to determine time conditions of the query because the size of time window is larger than normal time interval in query predicates. If it is too small, then the number of operations in a time window will be too small, so it is hard to find out accessing pattern of certain operations. An ideal window size must easily to find out the occurrence pattern of operations, in our DBroker system, time window size is one day length, that is to say we predict operations might arriving the next day. In a specific system user's behavior regular may be foreseen, window size can be determined at design time, while in a system that user's behavior can't be foreseen, window size should be determined based on statistical information of historical requests.

Query analyzer is implemented as a plug-in step of DBroker query mid-ware, and it is a phase between SQL parser and query optimizer. When a query hit in the prediction pool, then it will reconstruct the query plan, to make use of the speculative results.

C. Two Level Scoring Algorithm

Operation scoring algorithm is the basis of query prediction model. A regularly coming operation would be most likely to

arrival in the next time window. We use score to measure an operation arrival frequency in past windows, so high score means high frequency while low score means low frequency. If an operation keeps coming, then it will have a high increasing score, while a predicted operation failed to come in the following window, then its score will be cut by an invalid penalty.

We propose a two-level scoring (TLS) algorithm for query prediction, including score board and prediction pool two levels. Score board keeps the score of all operations, and prediction pool keeps all the speculative operations. The TLS algorithm scores the operations and then decides which operations to be chosen for prediction pool. Operations in prediction pool will be executed by speculative execution unit in advance, and then fill results entry into the prediction pool. Table II is the description of the TLS algorithm.

TABLE II. TWO-LEVEL SCORING ALGORITHM (TLS)

Input: op_window

Output: predict_pool

(1) For each op in score_board;

a) if op in op_window, then score_board[op].score += bonus;
b) else score_board[op].score -= penalty;
c) if score_board[op].score < score_board.threshold, then score_board.erase(op);

(2) For each op in op_window

if op not in score_board, then score_board.add[op]; /* *add new operations to score board with a initial score*/*

(3) predict_pool.clear(); /* *clear the prediction pool* */

(4) Choose high scored operations for prediction pool from score_board;

d) while predict_pool.size()< MAXSIZE
e) op=score_board.getnextTop(); /* *get the operation with the rest highest score*/*
f) if op.score >= predict_pool.threshhold, then predict_pool.add(op);
g) else break;

(5) Return predict_pool;

Unlike query analyzer analyzing user request as soon as it comes, scoring unit only refresh score board and prediction pool at the edge of two slide windows. In TLS algorithm, low score operations will be deleted from score board, to keep the score board in a moderate size, thus to reduce the algorithm overhead. Operations in prediction pool will be speculative executed in advance, if it fails to come in the following window, then the pre-execution effort is in vain. At running time, prediction pool does not always keep full, that's because operation score must higher than a threshold.

D. Speculative Execution

After predictor refreshing the content of prediction pool, query speculative execution unit pre-executes the operations, and then fill the results entry into the prediction pool. Speculative execution is much more complicated, and it needs to consider following two aspects problems.

The first problem is scheduling, including when does it to execute the operations, and how to schedule all the operations execution. The priority of speculative process must be lower than normal request. To make greatest use of system resource, common operations can be run as a batch. When a normal request comes and hit an operation under speculative execution, we introduce a forwarding technology to make use of result as early as possible. In Section 6 of this paper, we've just applied a naive scheduling method, which is to execute operations in prediction pool one after another as soon as predictor refreshes the pool.

The second problem is result management, including how to store the results and how long it would be kept. An operation result might be hit more than one times in the following window, or the result might be reused in more than one following window. An ideal scheme is that to store as many results to make most reuse of them as system capability permitted. Results management is not deeply studied in this paper, and in the experiment section, we use only a simple method, that is to keep speculative result for a window time.

In a cluster environment like DBroker, speculative execution can be implemented in two ways: global and local. Global way executes operations overall and collect all the results together to a query server node, while local method executes operations and keeps results separately in all the database nodes. If the speculative execution result size is relatively small, global way may be better, and if the result size is relatively large, local way could get high parallel degree in later processing.

V. COST ANALYSIS AND ADAPTIVE MECHANISM

A. Cost Analysis

Prediction and speculative execution may reduce query response time because of preparing in advance. We use T_{comp} representing for compile time, T_{opi} representing for time spent on executing operation op_i, and p_{inner} and $p_{pipeline}$ representing for inner and pipeline parallel degree respectively. In a system without query prediction, query response time is computed as follows:

$$T_{query} = T_{comp} + \left(\sum_i T_{opi} / p_{inner} \right) / p_{pipeline} \quad (1)$$

In a prediction system, we use 'a' representing for prediction accuracy rate, and then response time is computed as follows:

$$T_{query} = T_{comp} + \left((1-\alpha) \cdot \sum_i T_{opi} / p_{inner} \right) / p_{pipeline} \quad (2)$$

From the formula we can see that, the higher accuracy rate 'a' is, the lower query time T_{query} is.

If an operation in prediction pool failed to come in the next window, the speculative execution is in vain, that is a waste of system cost. So prediction model does not really decrease system total cost, on the contrary, it may introduce extra workload. In a system with prediction, operations will be executed only in two cases: a) they are in the prediction pool, and will be executed in advance; b) they are not in prediction pool, and will be executed at submitting time. We use 'ε' representing for efficiency rate and use the number of

operations executed for workload, then system workload and extra workload are computed as follows:

$$W = N_{predict} + N_{miss} = N_{op} \cdot \alpha / \varepsilon + N_{op} \cdot (1 - \alpha) = N_{op} \cdot \left(\frac{\alpha (1 - \varepsilon)}{\varepsilon} + 1 \right) \quad (3)$$

$$W_{extra} = N_{op} \cdot \alpha \cdot (1 - \varepsilon) / \varepsilon \quad (4)$$

In some particularly systems, the very same operations will arrive more than one time in a time window, so it can make reuse of speculative execution results. Suppose that operation repeated rate is '*r*' (in DBroker system it is 11.27%), and then system workload and extra workload are computed as follows:

$$W = N_{op} \cdot \left(\frac{\alpha (1 - r - \varepsilon)}{\varepsilon} + 1 \right) \quad (5)$$

$$W_{extra} = N_{op} \cdot \alpha \cdot (1 - r - \varepsilon) / \varepsilon \quad (6)$$

System workload reduces compared to system without operations repeating. In some cases, *(1-r- ε)* may be a minus value, then system workload will be less than original system. Generally, the higher accuracy rate '*a*' is, the more extra workload it will be introduced, while the higher efficiency rate '*ε*' is, the less extra workload is. Prediction model should make a balance between accuracy rate and efficiency rate.

B. Adaptive Mechanism

We've introduced an adaptive mechanism for automatically parameters adjusting. Prediction system parameters include score board threshold, prediction pool max size, prediction threshold, operation hit bonus and invalid penalty, etc. Adaptive mechanism consists of two components, system resource monitor and parameter adjuster.

- **System resource monitor**: used for system resource usage monitoring, then make an evaluation of current workload whether it is too heavy or too light or moderate.
- **Parameter adjuster**: used for collecting online predictor status, and adjusting parameters in a heuristic way based on cost and workload formula discussed in last subsection..

The purpose of adaptive mechanism is to make a balance between efficiency rate and accuracy rate, to make greatest improvement in query response time under system capability. Parameter adjusting is a heuristic and iterative course based on system tuning experience. When the system workload is too heavy in a period of time, then the adjuster will reduce the prediction pool size, or increase the prediction pool threshold, or increase the invalid penalty, all these means would increase the efficiency rate, as well as reduce the accuracy rate, and then system workload will be slowed down. In the other hand, when the system workload is too light, then the adjuster will increase the prediction pool size, or decrease the prediction pool threshold, or decrease the invalid penalty.

VI. EXPERIMENTS

A. Environment

We perform all experiments in a small scale 5-node cluster system, using the query SQL trace from DBroker system as user's query requests(more than 6000 queries), and the event stream data is also collected from DBroker system. The experimental environment consists of a query node(8*2.2G CPU, 4GB Memory, Linux AS4, 1Gb Ethernet), and 4 database nodes(4*2.4G CPU, 4GB Memory, Linux AS4, 5*146GB SCSI Disks, Oracle 10.2, 1Gb Ethernet). In a database node, one disk is used for system software, and the others are used for storing event streaming data. We make a comparison of our TLS algorithms and traditional cache replacing algorithms, and we also make a comparison of execution time between hitting and missing.

To illustrate the superiority of our TLS algorithm, we implemented two other algorithms, Naive and OLS methods, for comparison, and the two ideas are borrowed from cache and page replacing algorithms. In Naive method, operations occurred in recently several windows are speculative executed for the next window, and in OLS method, most recently used operations are predicted for next window.

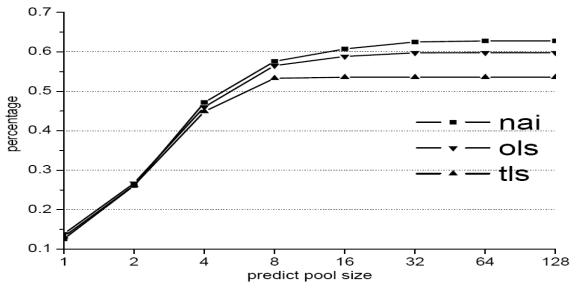

Figure 4. Accuracy rate in defferent methods

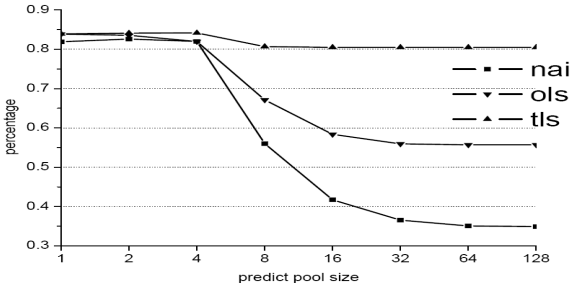

Figure 5. Efficiency rate in defferent methods

Fig. 4 and Fig. 5 show the comparison of the three methods while prediction pool max size is varying from 1 to 128. We can see that TLS method has a bit lower accuracy rate, but much higher efficiency rate than the other two methods. That's because naive method predicts as many operations as possible so it gets higher accuracy rate, OLS method does not consider invalid penalty and then gets lower efficiency rate. Fig. 6 shows the extra workload introduced to the system, it is computed from formula (5) in Section 5, while the operations repeated rate '*r*' is 11.27%. We can see that TLS has

introduced the least extra overhead, and much less than the other two.

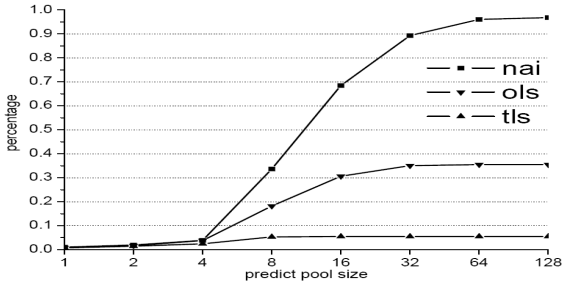

Figure 6. Extra worloadk of system in defferent methods

B. *Speculative Execution*

We also compare the query response time between hitting and missing. We've chosen 10 typical query requests from the query SQL trace, all of them query specific events data of previous day, and are I/O costing operations. Each query involves a large amount of data, about 4GB data in each database node, and the row count is about 50,000,000. So in all 4 nodes, each query may involve about 16GB data. We make a comparison of two speculative execution methods and no prediction way. Fig. 7 shows the execution time in different methods.

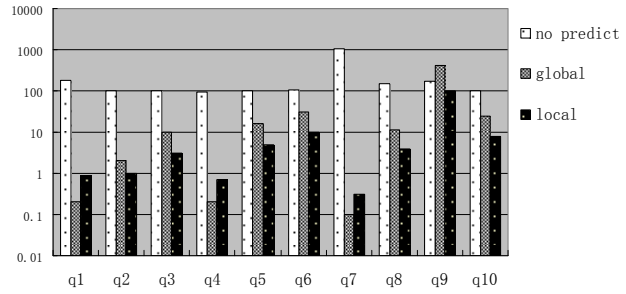

Figure 7. Query Response time in Different methods

From the figure we can see that in most cases, speculative execution can dramatically reduce response time, because of a greatly reduction of large amount of I/O cost. When speculative execution result size is small (as in q1, q4 and q7), the global method is better, but if the speculative execution result size is too large (as in q9), then global method may be even worse than no prediction method, that is because global method collected speculative execution result in one query server node, thus it can't develop large parallel degree in post processing. Local method is always better than no prediction method, a simple and effective way is to use local method.

VII. DISCUSSION

As an effective way to hide processing time, data prediction and speculative execution have attracted much more attentions in researchers, however to predict operations in event streaming is a novel concept. In a large scale data intensive system based on event streaming application, query processing time is dominated by I/O overhead. Hot-spot events accessing pattern gives an opportunity of prediction and speculative execution. We have developed an effective algorithm to predict table-scan and filter operations in future requests based on an analysis of historical query requests. Our model takes only table-scan and filter operations in account, partly because the operating data source is definite without dependency, so it is relatively easy to implement. Situations for other operations speculative execution are more complicated, and the operation data may depend on the output of earlier operations.

It is difficult to get both high accuracy rate and efficiency rate, naive method predict as many as possible without considering frequency so it can get a high accuracy rate, but efficiency rate is too much lower. OLS does not consider invalid penalty of prediction, and it also gets a poor efficiency. TLS method yields a little in accuracy rate by strict scoring algorithm, but gets a much higher efficiency rate. In a large scale system which is always under high workload, it is a useful and practical approach.

Our prediction model is implemented in the mid-ware layer of a cluster environment database system, and we've developed two methods of speculative execution, global and local. Generally speaking, global method is more suitable for small result size, which means this method is more suitable for high selectivity. A more effective way to choose for optimal method should be combined with system statistical information. Scheduling for speculative execution is not considered in this paper, actually it is a more difficult problem, especially in a cluster environment. To avoid repeated table scan cost in speculative execution, we can merged some operations together with a conjunction 'or', e.g. operation $\{col_1=x\}$ and operation $\{col_2=y\}$ can be merged as operation $\{col_1=x \text{ or } col_2=y\}$.

VIII. CONCLUSION AND FUTURE WORK

We've designed and implemented a query prediction model for large scale data intensive system based on event streaming application in this paper. Firstly we've made a study of query requests in DBroker system, a typical event streaming application system, and found hot-spot events accessing pattern. Then we've give a system design of query prediction model, we've developed TLS prediction algorithm as well as two speculative execution methods. Our experimental results demonstrate that the prediction algorithm can make a good balance in accuracy rate and efficiency rate, and our speculative execution method can get a great reduction of response time.

In the future, we will go on our research in two aspects. First we will expand hot-spot events prediction based on event rule detection, considering for event configuration information variation. Second we will go on researches on optimization for speculative execution, combined with database statistical information.

REFERENCES

[1] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model

and architecture for data stream management. VLDB, 12(2):120-139, August 2003.

[2] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In Proceedings of the 1996 ACM SIGMOD international conference, pages 137-146. SIGMOD, June 1996.

[3] F. Altiparmak, D. Chiu, and H. Ferhatosmanoglu. Incremental quantization for aging data streams. In Proceedings of the Seventh IEEE International Conference on Data Mining Workshops}, pages 527-532. ICDM, September 2007.

[4] R. Ananthakrishna, A. Das, J. Gehrke, F. Korn, S. Muthukrishnan, and D. Srivastava. Efficient approximation of correlated sums on data streams. IEEE Transactions on Knowledge and Data Engineering, 15(3):569-572, March 2003.

[5] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pages 1-16. PODS, June 2002.

[6] S. Babu and J. Widom. Continuous queries over data streams. ACM SIGMOD Record, 30(3):109-120, September 2001.

[7] G. Barish and C. A. Knoblock. Combining classification and transduction for value prediction in speculative plan execution. In Proceedings of IIWeb'2003, pages 131-136. IIWeb, August 2003.

[8] G. Barish and C. A. Knoblock. Speculative plan execution for information gathering. Artificial Intelligence, 172(4-5):413-453, March 2008.

[9] M. J. Carey, M. J. Franklin, M. Livny, and E. J. Shekita. Data caching tradeoffs in client-server dbms architectures. ACM SIGMOD Record}, 20(2):357 - 366, June 1991.

[10] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring streams: A new class of data management applications. In Proceedings of the 28th international conference on Very Large Data Bases, pages 215-226. VLDB, August 2002.

[11] S. Chandrasekara, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. Telegraphcq: continuous dataflow processing. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pages 668-668. CIDR, January 2003.

[12] S. Chandrasekaran and M. Franklin. Remembrance of streams past: overload-sensitive management of archived streams. In Proceedings of the 30th international conference on Very large data base, pages 348-359. VLDB, August 2004.

[13] S. Chandrasekaran and M. J. Franklin. Streaming queries over streaming data. In Proceedings of the 28th international conference on Very Large Data Bases}, pages 203-214. VLDB, August 2002.

[14] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagaracq: a scalable continuous query system for internet databases. In ACM SIGMOD Record, pages 379-390. SIGMOD, May 2000.

[15] R. L. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In Proceedings of the 1994 ACM SIGMOD international conference on Management of data, pages 150-160. SIGMOD, 1994.

[16] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data}, pages 40-51. SIGMOD, June 2003.

[17] P. M. Deshpande, K. Ramasamy, A. Shukla, and J. F. Naughton. Caching multidimensional queries using chunks. In Proceedings of the 1998 ACM SIGMOD international conference, pages 259-270. SIGMOD, June 1998.

[18] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In Proceedings of the 2002 ACM SIGMOD international conference, pages 61-72. SIGMOD, June 2002.

[19] S. Duan and S. Babu. Processing forecasting queries. In Proceedings of the 33rd international conference on Very large data bases, pages 711-722. VLDB, September 2007.

[20] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. ACM SIGMOD Record, 30(2):13-24, June 2001.

[21] J. G. D. Gooijer and R. J. Hyndman. 25 years of time series forecasting:a selective review. Tinbergen Institute Discussion Papers No. TI 05-068/4, June 2005.

[22] M. J., Franklin, B. Thor, Jonsson, and D. Kossmann. Performance tradeoffs for client-server query processing. ACM SIGMOD Record, 25(2):149-160, June 1996.

[23] F. Li, C. Chang, G. Kollios, and A. Bestavros. Characterizing and exploiting reference locality in data stream applications. In Proceedings of the 22nd International Conference on Data Engineering, page 81. ICDE, December 2006.

[24] S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In Proceedings of the 2002 ACM SIGMOD international conference, pages 49-60. SIGMOD, June 2002.

[25] B. Nag, P. M. Deshpande, and D. J. DeWitt. Using a knowledge cache for interactive discovery of association rules. In Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 244-253. SIGKDD, August 1999.

[26] N. Polyzotis and Y. Ioannidis. Speculative query processing. In Proceedings of the 2003 CIDR Conference. CIDR, January 2003.

[27] Q. Ren, M. H. Dunham, and V. Kumar. Semantic caching and query processing. IEEE Transactions on Knowledge and Data Engineering, 15(1):192-210, January 2003.

[28] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 426-435. SIGKDD, August 2003.

[29] U. Srivastava and J. Widom. Memory-limited execution of windowed stream joins. In Proceedings of the Thirtieth international conference on Very large data bases, pages 324-335. VLDB, August 2004.

[30] X. S. Wang, L. Gao, and M. Wang. Condition evaluation for speculative systems: a streaming time series case. In Proceedings of the Second Workshop on Spatio-Temporal Database Management, pages 65-72. STDBM, August 2004.

[31] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In Proceedings of the 2006 ACM SIGMOD international conference on Management of data, pages 407-418. SIGMOD, June 2006.

[32] D. Zhang, D. Gunopulos, V. J. Tsotras, and B. Seeger. Temporal aggregation over data streams using multiple granularities. In Extending Database Technology, pages 646-663. EDBT, March 2002.

[33] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope:A stream database for network applications. In Proceedings of the 2003 ACM SIGMOD international conference, June 2003.

[34] A. Y. Halevy. Answering queries using views: A survey. The VLDB Journal, 10(4), 2001.

[35] Roxana Geambasu, Tanya Bragin, Jaeyeon Jung, and Magdalena Balazinska. On-Demand View Materialization and Indexing for Network Forensic Analysis In Proc. of NetDB, April 2007.

[36] Sai wu, Jianzhong Li, Beng Chin Ooi, kian-Lee Tan. Just-in-time query retrieval over partially indexed data on structured P2P overlays. In Proceedings of the 2008 ACM SIGMOD international conference, June 2008.